Projet pour le cours du traitement de signal

Lefebvre Florian

May 27, 2024

Contents

1 Présentation du projet

1.1 Le cahier des charges

Le but de ce projet est de reconnaitre des petites balles de la taille d'une balle de Ping Pong de différentes couleurs avec une caméra. Pour réaliser ce projet, j'ai tout d'abord appris à utiliser OpenCV. Dans les deux premières parties de ce document, je présente des petits programmes qui m'ont ensuite servis d'outils pour développer l'application. Je termine ensuite par une petite application à partir de la reconnaissance de balles de couleur avec une caméra.

2 Les outils de base dans OpenCV

2.1 Présentation de la bibliothèque OpenCV

OpenCV (Open Source Computer Vision Library) est une bibliothèque open source de traitement d'image et de vision par ordinateur. Initialement développée par Intel en 1999, elle est désormais gérée par une communauté de développeurs. OpenCV est largement utilisée dans l'industrie, la recherche et les projets open source en raison de sa vaste gamme de fonctionnalités, de sa flexibilité et de sa facilité d'utilisation.

OpenCV a été créée en 1999 par Intel, dans le but de fournir une bibliothèque open source de traitement d'image et de vision par ordinateur. Elle était initialement développée en langage C, mais est désormais disponible dans de nombreux autres langages, telles que Python, C++, Java et MATLAB. Depuis sa création, OpenCV est devenue une bibliothèque de vision par ordinateur populaire et largement utilisée dans l'industrie, la recherche et les projets open source.

2.2 Un premier programme OpenCV

Ce premier programme permet d'afficher la même image avec des dimensions différentes. Voici le code commenté de ce premier exemple et le résultat de l'exécution du programme.

```
import cv2 as cv                    # import de la bibliotheque opencv

img = cv.imread("bouchon.png")      # lire l'image
```

```
print(img.shape)# afficher ses dimensions
print(img.size) # affiche la multiplication de h x l x p pixels
img_r = cv.resize(img, (300,420)) # redimentionner l'image
print(img_r.shape)
img_p = cv.resize(img, (0,0), fx=0.5, fy=0.7)# reduire l'image de 50%
en largeur et 70% en hautrur
print(img_p.shape)
cv.imshow("Image", img)                    #afficher la premiere image
cv.waitKey(0)                              # attente clavier
cv.destroyWindow("Image")          # liberation memoire de l'image
cv.imshow("Image_R", img_r)
cv.waitKey(0)
cv.destroyWindow("Image_R")
cv.imshow("Image_P", img_p)
cv.waitKey(0)                              # rester bloquer a l'infini
tant qu'on n'a pas
cv.destroyWindow("Image_P")        # appuyer sur une touche de clavier
```

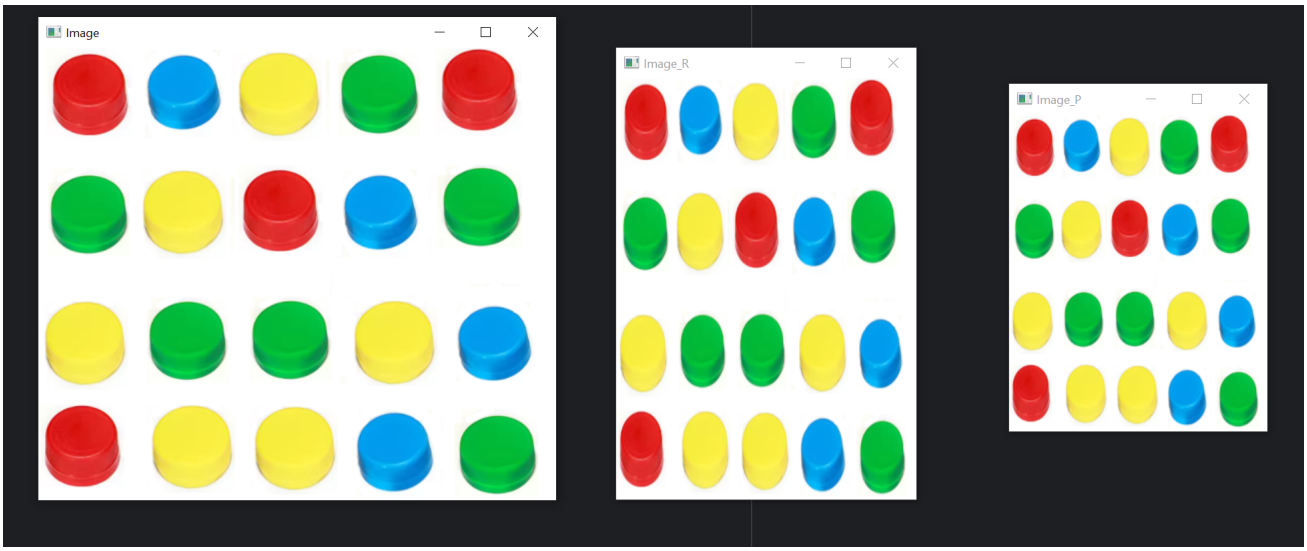

Figure 1: Afficher une image avec une réduction progressive

2.3 Récupérer une partie d'une image

L'objectif de ce programme est de récupérer une partie d'une image. Celle qui m'intéresse est le premier quart gauche de l'image.

```
import cv2 as cv
img = cv.imread("bouchon.png")
h,w,_ = img.shape # recuperer la largeur la hauteur et la profondeur
ROI = img[ : w//2 , : h//2] # des pixels et pas des nombres a virgule
cv.imshow("Image", img)
cv.imshow("ROI", ROI)  # ROI = region of interest
cv.waitKey(0)
```

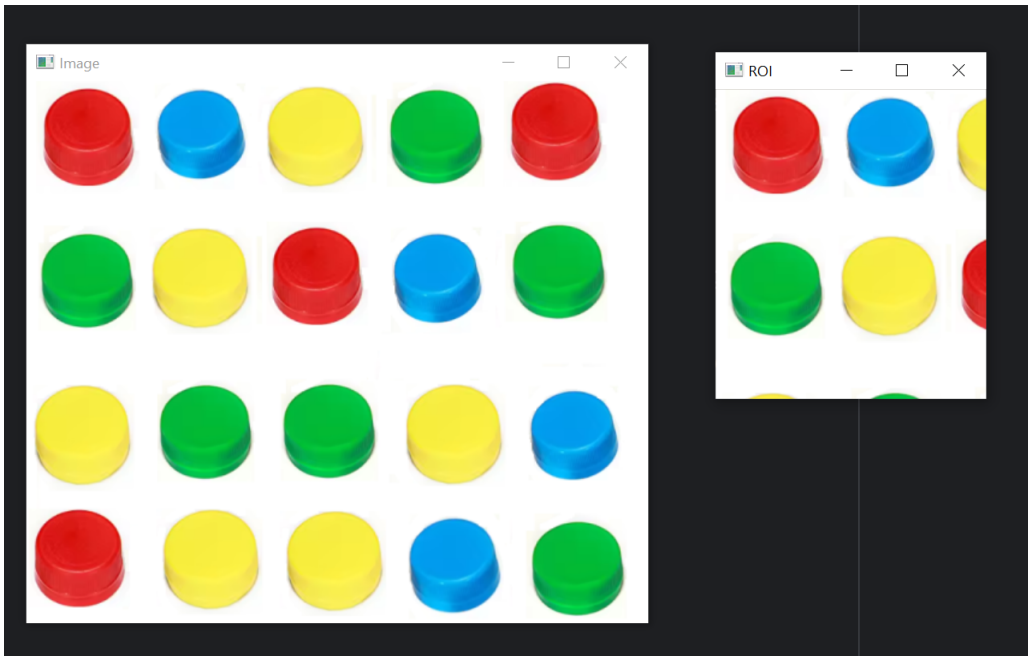

Figure 2: Récupérer une partie de l'image

2.4 Convertir une image en gris

Le but de ce progremme est de covertir une image en gris donc un seul plan. Ce programme a pour but de convertir une image en niveau de gris. La détection de ccontours est plus simple avec un seul plan et plus rapide.

```
import cv2 as cv
img = cv.imread("bouchon.png")
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)# BGR=bleu , vert , rouge
print(img_gray.shape)
cv.imshow("Image", img_gray)
cv.waitKey(0)
```

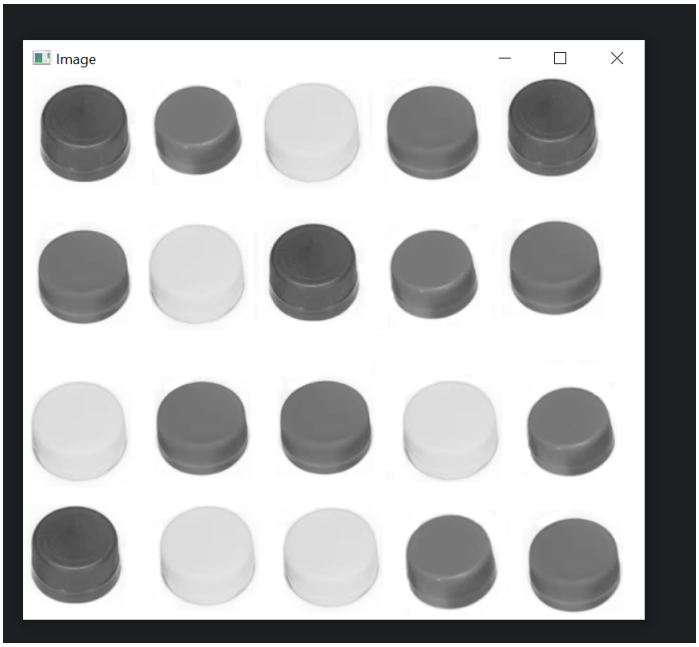

Figure 3: Une image convertie en gris

2.5 Rendre une image binaire

En partant d'une image qui contient des bactéries, on veut la rendre binaire (blanc ou noir). C'est à dire, on veut enlever les bruits de l'image. Pour cela, on va utiliser une fonction de seuillage.

```
import cv2 as cv
img = cv.imread("bacterie.png")
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
_, img_th = cv.threshold(img_gray,127,255,cv.THRESH_BINARY)#fonction
# de seuillage

# tout ce qui est au dessus de 127 aura la valeur 255 cad blanc
# et l'inverse sera noir
# THRESH_BINARY = le blanc reste le blanc et le noir reste le noir

print(_)
cv.imshow("Image", img_gray)
cv.imshow("img_th", img_th)
cv.waitKey(0)
```

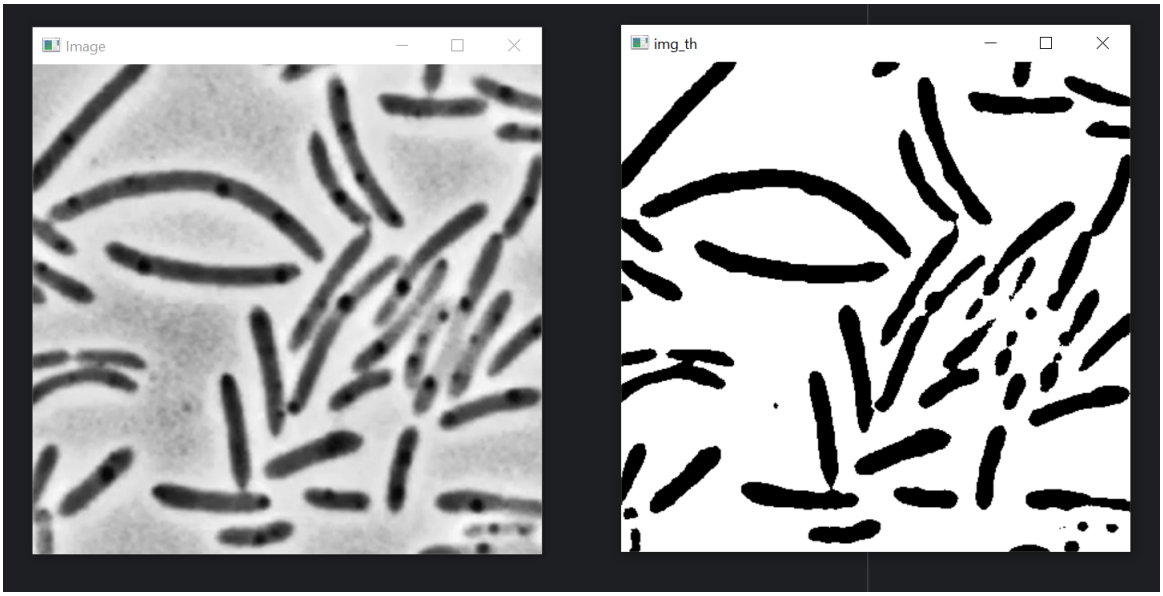

Figure 4: Rendre une image binaire

2.6 Utilisation de la caméra

Tout ce qu'on vient de voir avec les images restent valable avec la caméra.
Voici un premier exemple de programem qui utilise la caméra.

```
import cv2 as cv
camera = cv.VideoCapture(0) # camera de l'ordinateur
while 1:
    _, img = camera.read()#le premier argument est allume ou eteint
    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    cv.imshow('image', img)
    cv.imshow('img_gray', img_gray)
    if cv.waitKey(1) == 27: # touche ESC => on arrete la camera
        break
```

2.7 Affichage d'images côte à côte

Le but de ce programme est de pouvoir afficher des images ensemble l'une à côté d'une autre horizontalement ou verticalement voir même en mixant les deux.

```
import cv2 as cv
import numpy as np
img = cv.imread("bouchon.png")
```

```
img = cv.resize(img,(300,300))
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
_, img_th = cv.threshold(img_gray,127,255,cv.THRESH_BINARY)
img_gray = cv.cvtColor(img_gray,cv.COLOR_GRAY2BGR)
# pour rajouter les plans qu'il manque

img_th  = cv.cvtColor(img_th,cv.COLOR_GRAY2BGR)
img_f1= np.hstack((img,img_gray,img_th))
# on met les trois images horizotalement
img_f2= np.hstack((img_gray,img,img_th)) # idem
img_v = np.vstack((img_f1,img_f2))
# on met les deux rangees d'image du dessus verticalement
cv.imshow("Image", img_v)
cv.waitKey(0)
```

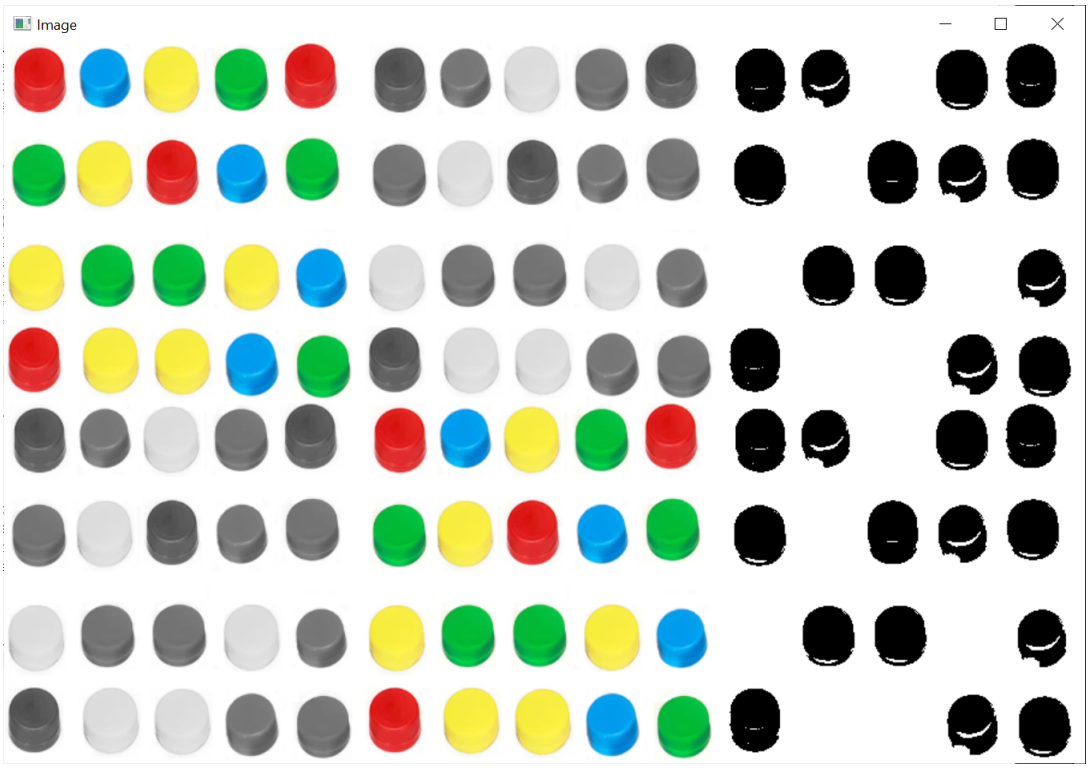

Figure 5: Affichage d'images côte à côte

2.8 Affichage d'une information sur une image

Le but de ce programme est de dessiner un rectangle dans une image, une ligne, un texte.

```
import cv2 as cv
import numpy as np
image = np.zeros((300,400,3))
#cree une image de 300 en hauteur sur 400 en largeur avec trois plans
# on met met ones a la place de zeros pour une matrice blanche
y,x,_ = image.shape
image[:,:] = 0,255,255
# pour opencv c'est BGR et non pas RGB
# ici on veut mettre une couleur dans la matrice
#image[30:50, :100] = 0,0,255
#ici on veut colorier une partie de l'image
cv.rectangle(image,(100,100), (200,200), (255,0,0),-1)
# image+point de depart
# +point d'arrivee+couleur+epaisseur-1 rect remplie
cv.circle(image,(100,50), 20, (0,255,0), 2)
# centre + rayon + couleur + epaisseur
cv.line(image,(0,0), (x//2,y//2), (255,255,255), 4)
cv.putText(image,"Opencv",(150,200),
           cv.FONT_HERSHEY_COMPLEX,1.5,(0,0,0),1)
# chaine + position + fonte + echelle + couleur + epaisseur
cv.imshow("Mon_image",image)
cv.waitKey(0)
```

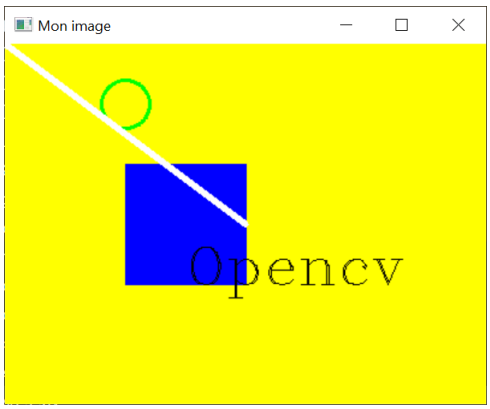

Figure 6: Affichage d'informations sur une image

3 Détection des contours et des couleurs

3.1 Détection de contour très mauvais

Le but du programme ici est de faire une détection de contour avec la fonction *Canny*. On remarque qu'elle est de très mauvaise qualité. Canny préconise une image en gris car si on utilise plusieurs plans, l'ordinateur mettra plus de temps de calcul. On constate en effet que si on passe à la fonction *Canny* l'image en gris, le temps de réponse est plus rapide. *Canny* calcule le gradient selon plusieurs directions. Quand le gradient est fort, il trace une ligne représentant un contour. Quand le gradient est faible, il ne trace pas de contour comme le montre les figures 7, 8 et 9.

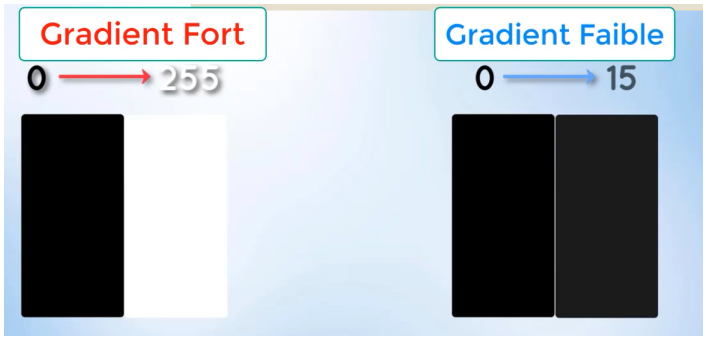

Figure 7: gradient fort et faible

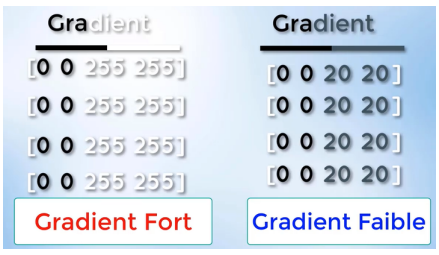

Figure 8: gradient fort et faible

Notre matrice d'image est une matrice de valeur de zéro à 255. La figure 10 montre le résultat d'une détection de contour sur l'image de la tête d'un éléphant. On voit que cette détection de contour avec la fonction *Canny* est imparfaite car elle détecte trop de contours.

```
import cv2 as cv
import numpy as np
```

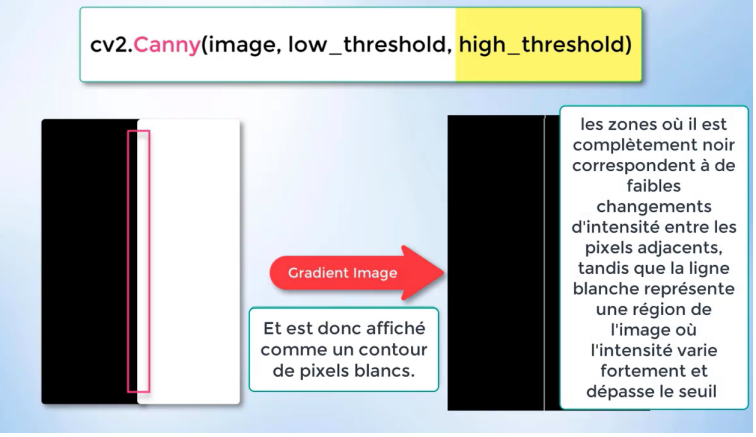

Figure 9: gradient fort et faible

```
img = cv.imread("elephant.png")
img = cv.resize(img, (400, 400))
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
img_canny = cv.Canny(img, 60, 120)
cv.imshow("Image", img_canny)
cv.waitKey(0)
```

Figure 10: Image d'un éléphant

3.2 Utilisation d'un filtre

Le but d'un filtre est d'éliminer les pixels indésirables qui sont du bruit comme le montre l'image suivante. Le but du filtre va être de lisser une image. Si un pixel ne correspond pas à ses voisins, on va lui donner la moyenne de ses voisins. La figure suivante montre un pixel qui n'est pas comme ses voisins. Ce pixel va créer du bruit. Le filtre de Gauss va faire la somme de tous les nombres autour et mettre la moyenne. Ce filtre doit avoir une valeur impair,

Figure 11: Détection de contours avec la fonction Canny

Figure 12: Le lissage d'une image avec un filtre

soit 3, 5, 7, 9 et ainsi il se déplace de gauche vers la droite pas à pas et après du haut vers le bas. Pour faire ce filtre, on doit lui donner une image au niveau du gris. Si le noyau est petit comme 3, il y aura plus de précision mais l'ordinateur mettra plus de temps. Dans la figure 13, on voit un point qui vaut 90, et qui est différent de ses voisins. On lui affectera la moyenne de ses voisins.

Voici le résultat de la reconnaissance de détours mais cette fois-i en intégrant le filtre de Gauss. On voit que le résultat est nettement plus satisfaisant.

3.3 Le passage à une caméra

Voici un programme qui récupère les images d'une caméra et qui applique la fonction Canny avec un filtre de Gauss.

```
import cv2 as cv
```

Figure 13: Le calcul d'un lissage

Figure 14: Intégration du filtre de Gauss

```
import numpy as np

cap = cv.VideoCapture(0)

while 1:
    _, img = cap.read()
    img = cv.resize(img, (400, 400))
    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    img_blur = cv.GaussianBlur(img_gray, (7, 7), 0)
    img_canny = cv.Canny(img_blur, 60, 120)

    img_f = np.hstack((img_gray, img_blur, img_canny))

    cv.imshow('image', img_f)
    if cv.waitKey(1) == 27:
```

```
        break
```

3.4 Comment créer un trackbar ?

Le but de ce programme est de montrer comment créer un trackbar pour pouvoir dessiner un carré dont la dimension dépend de la valeur de l'échelle sélectionnée.

```
import cv2 as cv
import numpy as np


def vide(a): # fonction appelee par trackbar
    pass


img = np.zeros((300, 400, 3))#creer une matrice avec une couleur noir

cv.namedWindow('Trackbar') # creer une fenetre avec un nom
cv.resizeWindow('Trackbar', 400, 300)

cv.createTrackbar('valeur', 'Trackbar', 0, 255, vide)

while 1:
    x = cv.getTrackbarPos('valeur', 'Trackbar')

    img[:x, :x] = x, 0, 0
    cv.imshow('Trackbar', img)

    if cv.waitKey(1) == 27:
        break
```

3.5 Application des trackbar

Le programme suivant permet d'utiliser les trackbars pour paramétrer le filtre de Gauss, et l'algorithme Canny. Ainsi, il est possible de tester plus facilement les paramètres pour voir la meilleure détection de contour.

```
import cv2 as cv
import numpy as np
```

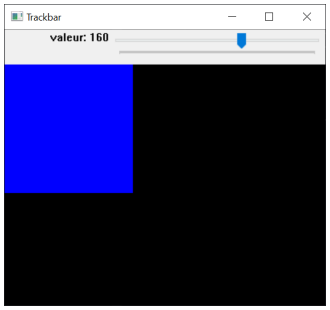

Figure 15: Gestion d'un trackbar

```
def vide(a):
    pass


cv.namedWindow('Trackbar')
cv.resizeWindow('Trackbar', 300, 300)

cv.createTrackbar('Ksize', 'Trackbar', 3, 40, vide)
# taille du noyau du filtre
cv.createTrackbar('canny_min', 'Trackbar', 0, 255, vide)
cv.createTrackbar('canny_max', 'Trackbar', 255, 255, vide)

cap = cv.VideoCapture(0)

while 1:
    _, img = cap.read()
    img = cv.flip(img, 1) # inverser l'image selon l'axe x
    img = cv.resize(img, (400, 400))
    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    filtre = cv.getTrackbarPos('Ksize', 'Trackbar')
    canny_min = cv.getTrackbarPos('canny_min', 'Trackbar')
    canny_max = cv.getTrackbarPos('canny_max', 'Trackbar')

    if filtre % 2 != 0:
        new_filtre = filtre
```

```
img_blur = cv.GaussianBlur(img_gray, (new_filtre, new_filtre), 0)
img_canny = cv.Canny(img_blur, canny_min, canny_max)
img_f = np.hstack((img_gray, img_blur, img_canny))
cv.imshow('Trackbar', img_f)
if cv.waitKey(1) == 27:
    break
```

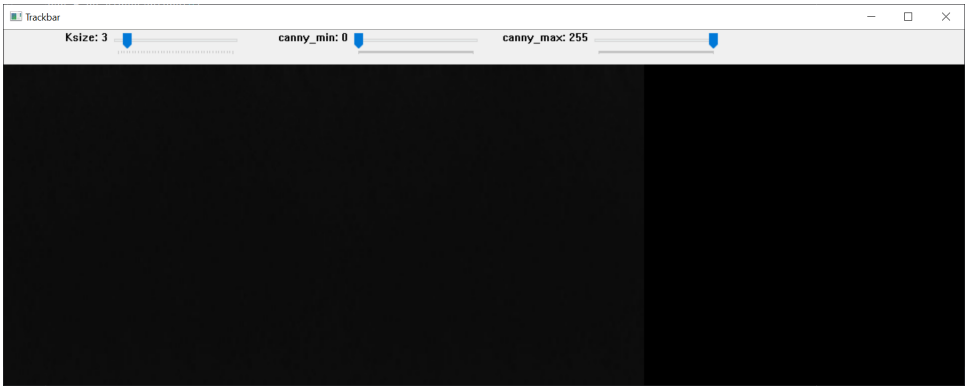

Figure 16: Utilisation de trackbars pour paramétrer le filtre de Gauss et Canny

3.6 La détection des contours

Les contours sont des lignes ou des courbes qui délimitent ou couvrent la limite complète d'un objet dans une image. Ils sont importants pour la détection des objets ou l'analyse de forme. OpenCV stocke les contours dans une liste de listes comme le montre la figure 17.

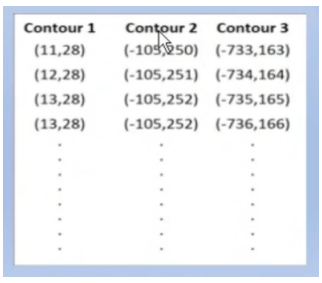

Figure 17: La détection de contours dans openCV

Voici dans la figure 18 les détails de gestion de cette fonction. Cette fonction prend une image binaire de 0 à 255 qu'on obtient grâce à la fonction

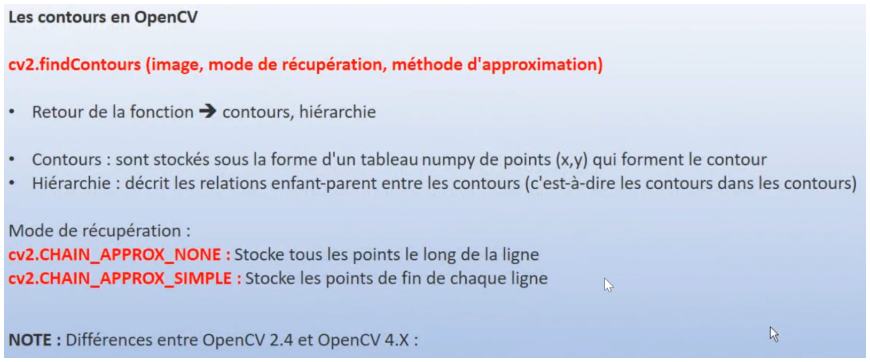

Figure 18: Les paramètres de la fonction de gestion de contour dans openCV

Canny. Le mode de récupération permet de récupérer tous les pixels et leurs coordonnées ou bien juste leur extrémité. La méthode d'approximation s'appuie sur la hiérarchie des contours présentée dans la figure 19. Ensuite

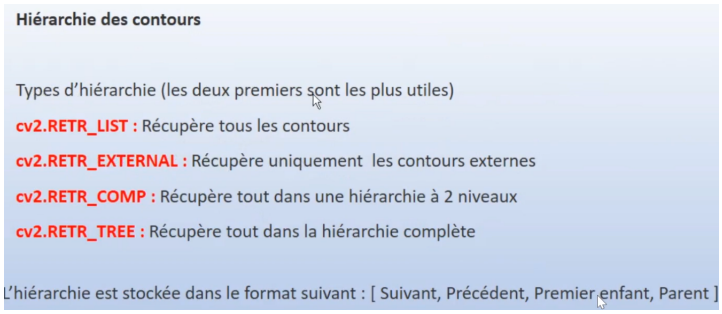

Figure 19: La hiérarchie des contours dans openCV

la fonction *drawContours* va permettre de dessiner les contours de l'étape précédente comme le montre la figure 20. Le paramètre *image* est l'image

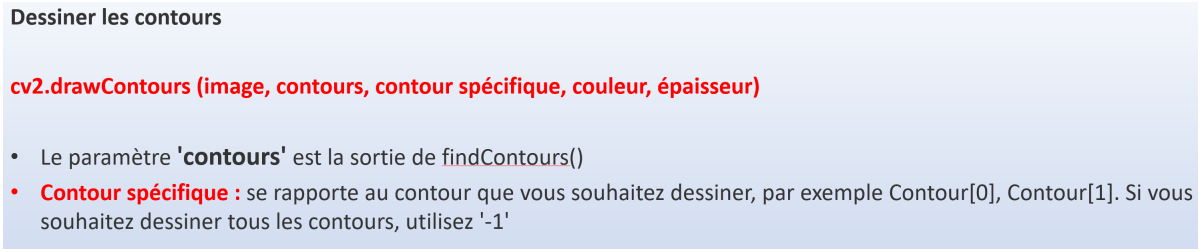

Figure 20: Dessiner les contours dans openCV

d'entrée. Le paramètre *contours* est la liste des coordonnées en x et y des contours par la méthode *findContours()*. La méthode *approxPolyDP* a pour

objecfif de corriger de légères distorsions d'un contour. Elle va en fait réduire le nombre de sommets. La documentation de cette fonction se trouve dans la figure 21.

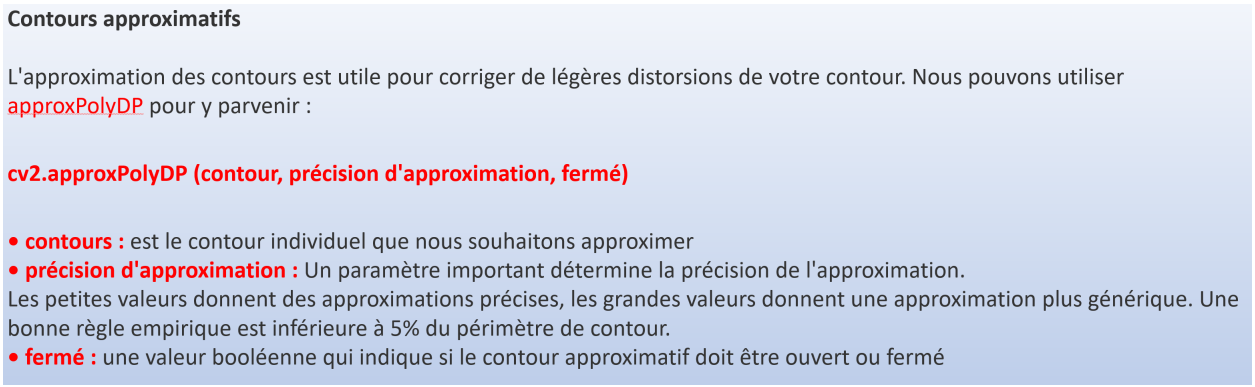

Figure 21: Corrections d'approximation

Voici un programme qui permet de reconnaitre des types de formes dans un ensemble d'objets géométriques. Le résultat de ce programme se trouve dans la figure 22.

```
import cv2 as cv

text_form = ''

img = cv.imread('contour.png')
img = cv.resize(img, None, fx=0.8, fy=0.8)

img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
# obtenir un seul plan pour un calcul rapide
img_blur = cv.GaussianBlur(img_gray, (3, 3), 0)
# utilisation d'un filtre
img_canny = cv.Canny(img_blur, 10, 40)
# obtenir une image binaire

contour, _ = cv.findContours(img_canny,
            cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
# que les contours externes
# et on stocke tous les points des contours
for cnt in contour:
    cv.drawContours(img, cnt, -1, (255, 0, 0), 3)
    area = cv.contourArea(cnt) # calcul de la surface
```

```
        peri = cv.arcLength(cnt, True) # calcul du perimetre

        approx = cv.approxPolyDP(cnt, 0.03 * peri, True)
        # 0.03 = 3 % but minimiser le nb de sommets
        x, y, w, h = cv.boundingRect(approx)

        cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
        # ainsi on peut dessiner un rectangle autour de l'objet

        nb_point = len(approx) # on calcule le nombre de sommets

        # print(nb_point)

        if nb_point == 3:
            text_form = "Tri"

        elif nb_point == 4:
            dist = w / h
            if dist > 0.95 and dist < 1.05:
                text_form = "Carre"
            else:
                text_form = "Rect"
        else:
            text_form = "Cercle"

        cv.putText(img, text_form, (x, y - 10),
            cv.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0), 1)

cv.imshow("Image", img)
cv.waitKey(0)
```

On adapte le programme à l'utilisation de la caméra.

```
import cv2 as cv


def vide(a):
    pass


cv.namedWindow('Trackbar')
cv.resizeWindow('Trackbar', 600, 300)
```

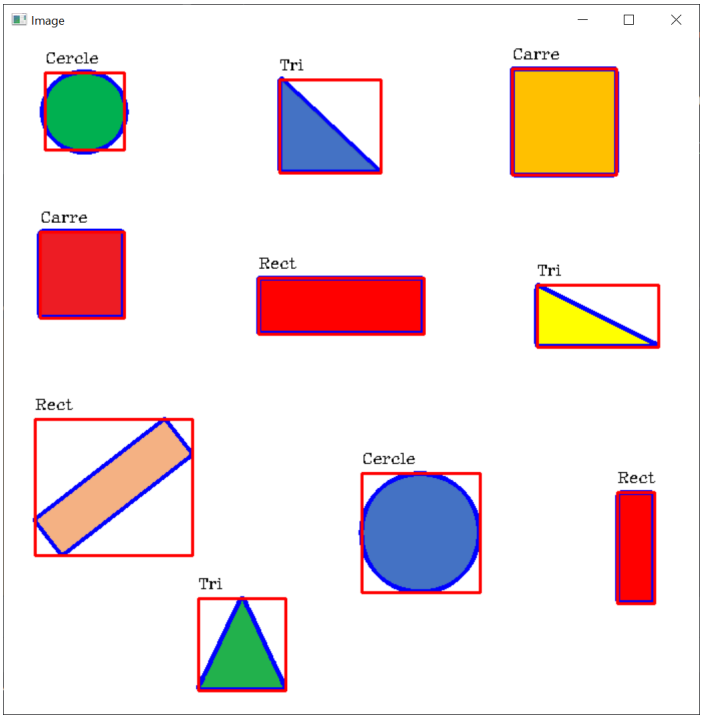

Figure 22: Reconnaissance de formes géométriques

```
cv.createTrackbar('Ksize', 'Trackbar', 3, 40, vide)
cv.createTrackbar('canny_min', 'Trackbar', 0, 255, vide)
cv.createTrackbar('canny_max', 'Trackbar', 255, 255, vide)
cv.createTrackbar('area', 'Trackbar', 500, 2000, vide)

cap = cv.VideoCapture(0)  # 0 pour la camera interne de l'ordinateur

while 1:
    _, img = cap.read()
    img = cv.flip(img, 1)
    # img = cv.resize(img, (400, 400))
    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    filtre = cv.getTrackbarPos('Ksize', 'Trackbar')
    canny_min = cv.getTrackbarPos('canny_min', 'Trackbar')
    canny_max = cv.getTrackbarPos('canny_max', 'Trackbar')
    area_max = cv.getTrackbarPos('area', 'Trackbar')

    if filtre % 2 != 0:
        new_filtre = filtre
```

```
    img_blur = cv.GaussianBlur(img_gray,
            (new_filtre, new_filtre), 0)
img_canny = cv.Canny(img_blur, canny_min, canny_max)

contour, _ = cv.findContours(img_canny,
     cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)

for cnt in contour:
    area = cv.contourArea(cnt)

    if area > area_max:
        cv.drawContours(img, cnt, -1, (255, 0, 0), 3)
        peri = cv.arcLength(cnt, True)

        approx = cv.approxPolyDP(cnt, 0.03 * peri, True)
        x, y, w, h = cv.boundingRect(approx)
        cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
        nb_point = len(approx)

        if nb_point == 3:
            text_form = "Tri"

        elif nb_point == 4:
            dist = w / h
            if dist > 0.95 and dist < 1.05:
                text_form = "Carre"
            else:
                text_form = "Rect"
        else:
            text_form = "Cercle"

        cv.putText(img, text_form, (x, y - 10),
            cv.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0), 1)

key = cv.waitKey(1)
cv.imshow('Trackbar', img)
if key == 27:
    break
```

3.7 La détection des couleurs

La figure 23 présente le format par défaut des images lus par OpenCV. L'image est représentée avec trois canaux : bleu, vert, et rouge. On va convertir cette image dans le format HSV.

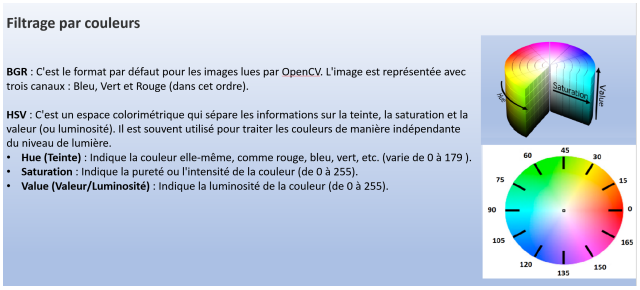

Figure 23: La fonction pour effectuer un filtrage

La fonction qu'on va utiliser est la fonction inRange qui va permettre d'effectuer des opérations de seuillage. Elle est présentée dans la figure 24.

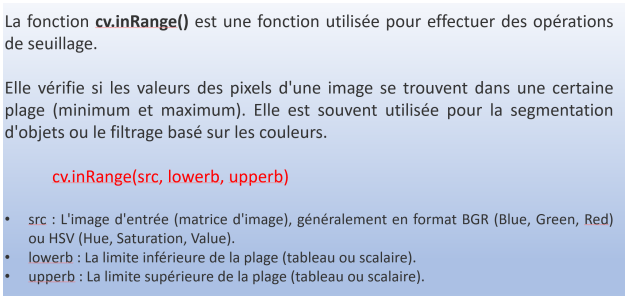

Figure 24: Filtrage par couleurs

Le premier programme convertit tout d'abord l'image avec les valeurs de HSV. On va créer des trackbars pour parammétrer HSV permettant ainsi de sélectionner une couleur. Il va en falloir 6. La première va de 0 à 179. Pour le Hue max, la valeur ira de 179 à 0. La saturation ira de 0 à 255. Pour les valeurs maximales, elles commenceront à 255. On va ensuite créer un filtre avec la fonction *cvinRange*. La figure 25 montre la sélection du jaune parmi l'ens

```
import cv2 as cv
import numpy as np
def vide(a):
    pass
cv.namedWindow("Trackbar")
```

```
cv.resizeWindow("Trackbar",600,250)
cv.createTrackbar("Hue_min","Trackbar",0,179,vide)
cv.createTrackbar("Hue_max","Trackbar",179,179,vide)
cv.createTrackbar("Sat_min","Trackbar",0,255,vide)
cv.createTrackbar("Sat_max","Trackbar",255,255,vide)
cv.createTrackbar("Val_min","Trackbar",0,255,vide)
cv.createTrackbar("Val_max","Trackbar",255,255,vide)
img = cv.imread("bouchon.png")
img = cv.resize(img,(300,300))
hsv_img = cv.cvtColor(img,cv.COLOR_BGR2HSV) # onvertit l'image au form
while True:
    h_min = cv.getTrackbarPos("Hue_min","Trackbar")
    # recuperation de l'information
    s_min = cv.getTrackbarPos("Sat_min", "Trackbar")
    v_min = cv.getTrackbarPos("Val_min", "Trackbar")
    h_max = cv.getTrackbarPos("Hue_max","Trackbar")
    s_max = cv.getTrackbarPos("Sat_max", "Trackbar")
    v_max = cv.getTrackbarPos("Val_max", "Trackbar")
    hsv_min = np.array([h_min,s_min,v_min])
    # creer deux vecteurs minimal et maximal pour simplifier
    hsv_max = np.array([h_max, s_max, v_max])
    mask = cv.inRange(hsv_img,hsv_min,hsv_max)
    # les pixels qui seront dans cette plage seront blanc : 255 et
    # le reste aura la valeur 0
    img_color = cv.bitwise_and(img,img,mask=mask)
    mask = cv.cvtColor(mask,cv.COLOR_GRAY2BGR)
    img1 = np.hstack((img,hsv_img))
    img2 = np.hstack((mask,img_color))
    img_final = np.vstack((img1,img2))
    cv.imshow("Image", img_final)
    if cv.waitKey(1) == 27:
        break
```

On va ensuite faire un *and* entre l'image d'origine et l'image en noir et blanc grâce à la fonction *bitwise* dont la définition est dans la figure 26.

En haut à gauche, on aura l'image d'origine, à droite l'image HSV, en bas à gauche le masque et en bas à droite le résultat. Le masque binaire est un seul plan. Il faudra le convertir en trois plans pour que cet assemblage puisse se faire.

```
import cv2 as cv
import numpy as np
```

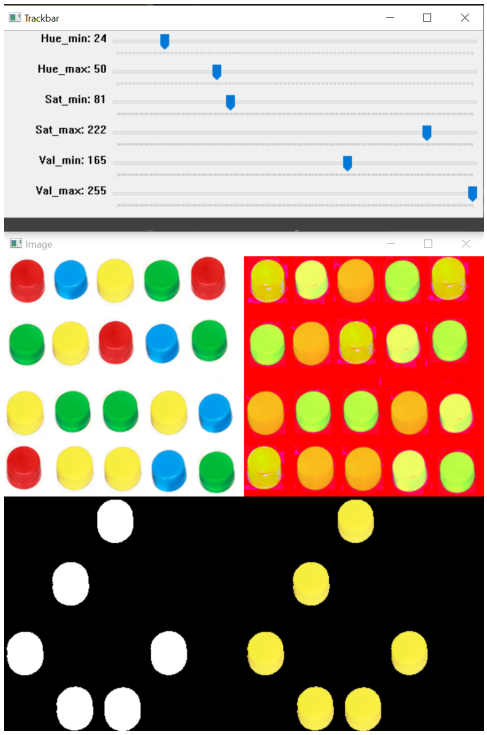

Figure 25: La sélection du jaune

La fonction cv.bitwise() fournit plusieurs fonctions bit à bit (bitwise) pour manipuler les images au niveau binaire. Ces fonctions permettent de faire des opérations logiques telles que ET, OU, NON et OU exclusif sur les images. Voici les principales fonctions bit à bit :

- result = cv.bitwise_and(image1, image2, mask)
- result = cv.bitwise_or(image1, image2, mask)
- result = cv.bitwise_xor(image1, image2, mask)
- result = cv.bitwise_not(image, mask)

cv.bitwise_and() extrait la partie de l'image dans la région définie par le masque.

Figure 26: La fonction bitwise

```
def vide(a):
```

Figure 27: Implémentation d'un filtre de couleur

```
    pass
cv.namedWindow("Trackbar")
cv.resizeWindow("Trackbar",600,250)
cv.createTrackbar("Hue_min","Trackbar",0,179,vide)
cv.createTrackbar("Hue_max","Trackbar",179,179,vide)
cv.createTrackbar("Sat_min","Trackbar",0,255,vide)
cv.createTrackbar("Sat_max","Trackbar",255,255,vide)
cv.createTrackbar("Val_min","Trackbar",0,255,vide)
cv.createTrackbar("Val_max","Trackbar",255,255,vide)
cap = cv.VideoCapture(0)
while True:
    _, img = cap.read()
    img = cv.flip(img,1)
    img = cv.resize(img, (300, 300))
    hsv_img = cv.cvtColor(img, cv.COLOR_BGR2HSV)
    h_min = cv.getTrackbarPos("Hue_min","Trackbar")
    s_min = cv.getTrackbarPos("Sat_min", "Trackbar")
    v_min = cv.getTrackbarPos("Val_min", "Trackbar")
    h_max = cv.getTrackbarPos("Hue_max","Trackbar")
    s_max = cv.getTrackbarPos("Sat_max", "Trackbar")
    v_max = cv.getTrackbarPos("Val_max", "Trackbar")
```

```
hsv_min = np.array([h_min,s_min,v_min])
hsv_max = np.array([h_max, s_max, v_max])
mask = cv.inRange(hsv_img,hsv_min,hsv_max)
img_color = cv.bitwise_and(img,img,mask=mask)
mask = cv.cvtColor(mask,cv.COLOR_GRAY2BGR)
img1 = np.hstack((img,hsv_img))
img2 = np.hstack((mask,img_color))
img_final = np.vstack((img1,img2))
cv.imshow("Image", img_final)
if cv.waitKey(1) == 27:
    break
```

4 La conception du projet

4.1 La sauvegarde des paramètres

Le premier programme consiste à sauvegarder les paramètres des couleurs que je souhaite reconnaître. Si je tape le caractère 'j', je sauvegarderai le jaune par exemple. Cela se fera avec la fonction save de Numpy. Il va falloir stocker *hsv_min* et *hsv_max*. Je stockerai ces deux paramètres dans une liste avant de la sauvegarder dans un fichier qui s'appelle **col_jaune**. Je peux faire la même chose pour d'autres couleurs. La figure 28 présente la sauvegarde des paramètres de la couleur jaune.

```
import cv2 as cv
import numpy as np


def vide(a):
    pass


cv.namedWindow("Trackbar")
cv.resizeWindow("Trackbar", 600, 250)

cv.createTrackbar("Hue_min", "Trackbar", 0, 179, vide)
cv.createTrackbar("Hue_max", "Trackbar", 179, 179, vide)
cv.createTrackbar("Sat_min", "Trackbar", 0, 255, vide)
cv.createTrackbar("Sat_max", "Trackbar", 255, 255, vide)
cv.createTrackbar("Val_min", "Trackbar", 0, 255, vide)
cv.createTrackbar("Val_max", "Trackbar", 255, 255, vide)
```

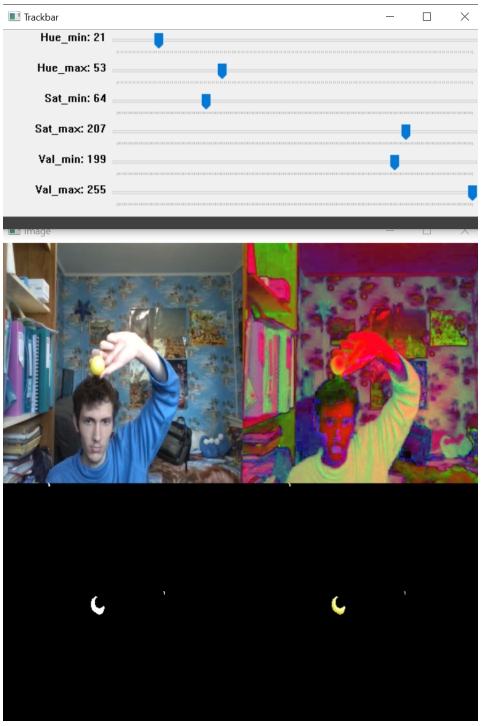

Figure 28: Sauvegarde de la couleur jaune

```
cap = cv.VideoCapture(0)

while True:
    _, img = cap.read()
    img = cv.flip(img, 1)
    img = cv.resize(img, (300, 300))
    hsv_img = cv.cvtColor(img, cv.COLOR_BGR2HSV)

    h_min = cv.getTrackbarPos("Hue_min", "Trackbar")
    s_min = cv.getTrackbarPos("Sat_min", "Trackbar")
    v_min = cv.getTrackbarPos("Val_min", "Trackbar")

    h_max = cv.getTrackbarPos("Hue_max", "Trackbar")
    s_max = cv.getTrackbarPos("Sat_max", "Trackbar")
    v_max = cv.getTrackbarPos("Val_max", "Trackbar")

    hsv_min = np.array([h_min, s_min, v_min])
    hsv_max = np.array([h_max, s_max, v_max])
```

```
        mask = cv.inRange(hsv_img, hsv_min, hsv_max)
        img_color = cv.bitwise_and(img, img, mask=mask)

        mask = cv.cvtColor(mask, cv.COLOR_GRAY2BGR)

        img1 = np.hstack((img, hsv_img))
        img2 = np.hstack((mask, img_color))
        img_final = np.vstack((img1, img2))

        cv.imshow("Image", img_final)

        key = cv.waitKey(1)

        if key == 27:
            break

        if key == ord("j"):
            HSV = [hsv_min, hsv_max]
            np.save("col_jaune", HSV)
            print(HSV)

        if key == ord("r"):
            HSV = [hsv_min, hsv_max]
            np.save("col_rouge", HSV)
            print(HSV)

cap.release()                                   # on arrete la camera
cv.destroyAllWindows()
```

4.2 La reconnaissance des balles jaune et rouge

Le programme va devoir au début charger les deux matrices, celle du jaune et celle du rouge. Il va allumer la caméra. Il va falloir ensuite créer le masque pour chaque couleur en exploitant les paramètres qu'on a sauvegardé. Ensuite grâce à la fonction findContours, on va détecter les contours externes à partir du masque et tous les points. On obtient ainsi une de tous les coordonnées de chaque contour. Ensuite, on balaye tous ces contours en affichant un petit rectangle. Voici le programme correspondant à la reconnaissance du jaune et du rouge.

```
HSV = np.load('col_jaune.npy')
hsv_min = HSV[0]
hsv_max = HSV[1]

HSV_R = np.load('col_rouge.npy')
hsv_min_R = HSV_R[0]
hsv_max_R = HSV_R[1]

cap = cv.VideoCapture(0)

while 1:
    _, img = cap.read()
    img = cv.flip(img, 1)
    img = cv.resize(img, (300, 300))
    hsv_img = cv.cvtColor(img, cv.COLOR_BGR2HSV)

    mask = cv.inRange(hsv_img, hsv_min, hsv_max)
    img_color = cv.bitwise_and(img, img, mask=mask)
    mask_rouge = cv.inRange(hsv_img, hsv_min_R, hsv_max_R)
    img_rouge = cv.bitwise_and(img, img, mask=mask_rouge)
    cnt_1,_ = cv.findContours(mask,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
    cnt_2,_ = cv.findContours(mask_rouge,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
    for contour in cnt_1:
        x, y, w, h = cv.boundingRect(contour)
        cv.rectangle(img, (x, y), (x + w, y + h), (0, 255, 255), 2)
    for contour in cnt_2:
        x, y, w, h = cv.boundingRect(contour)
        cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv.imshow("Image", img)
    key = cv.waitKey(1)
    if key == 27:
        break
cap.release()
cv.destroyAllWindows()
```

Le programme suivant permet de reconnaitre uniquement la couleur jaune en rajoutant la fonction d'approximation dec contours.

```
import cv2 as cv
import numpy as np
```

```
HSV = np.load('col_jaune.npy')

hsv_min = HSV[0]
hsv_max = HSV[1]

cap = cv.VideoCapture(0)

while 1:
    _, img = cap.read()
    img = cv.flip(img, 1)
    img = cv.resize(img, (300, 300))
    hsv_img = cv.cvtColor(img, cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv_img, hsv_min, hsv_max)
    img_color = cv.bitwise_and(img, img, mask=mask)
    contour, _ = cv.findContours(mask,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
    for cnt in contour:
        area = cv.contourArea(cnt)
        if area > 200:
            # cv.drawContours(img,cnt,-1,(0,255,0),2)

            peri = cv.arcLength(cnt, True)
            approx = cv.approxPolyDP(cnt, 0.05 * peri, True)
            x, y, w, h = cv.boundingRect(approx)
            cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
            # cv.imshow("MASK", mask)
    cv.imshow("Image", img)
    # cv.imshow("bitwise",img_color)

    key = cv.waitKey(1)
    if key == 27:
        break
cap.release()
cv.destroyAllWindows()
```

4.3 Un exemple d'application

Le but de ce programme est de contrôle le volume de son ordinateur avec une balle. On va tracer une ligne verticalement. Si la balle est dans une zone,

j'augmente le volume de mon ordinateur. Si la balle est dans l'autre zone, je diminue le volume de l'ordinateur. Pour cela, j'utilise une bibliothèque qui s'appelle *pyautogui* qui me permet de contrôler le volume du son avec le clavier.

Voici le code de l'application :

```
import cv2 as cv
import numpy as np
import pyautogui

HSV = np.load('col_jaune.npy')

hsv_min = HSV[0]
hsv_max = HSV[1]

cap = cv.VideoCapture(0)

while 1:

    _, img = cap.read()
    img = cv.flip(img, 1)
    img = cv.resize(img, (300, 300))
    hsv_img = cv.cvtColor(img, cv.COLOR_BGR2HSV)

    mask = cv.inRange(hsv_img, hsv_min, hsv_max)
    img_color = cv.bitwise_and(img, img, mask=mask)

    contour, _ = cv.findContours(mask,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)

    for cnt in contour:
        area = cv.contourArea(cnt)
        if area > 200:

            peri = cv.arcLength(cnt, True)
            approx = cv.approxPolyDP(cnt, 0.05 * peri, True)
            x, y, w, h = cv.boundingRect(approx)
            cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)


            if x > 100:
```

```
                pyautogui.press('volumeup')
            else:
                pyautogui.press('volumedown')

    cv.line(img, (100, 0), (100, img.shape[0]), (0, 0, 0), 3)

    cv.imshow("Image", img)

    key = cv.waitKey(1)
    if key == 27:
        break
cap.release()
cv.destroyAllWindows()
```

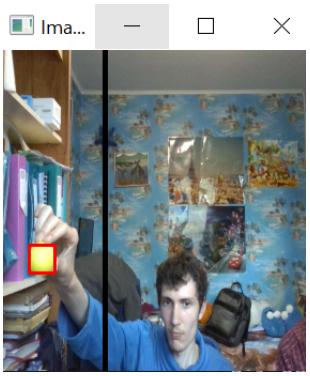

Figure 29: Contrôle du volume à l'aide de la balle jaune

Figure 30: Vue du volume sonore

5 Conclusions

Ce projet est une première expérience dans le domaine de la vision par ordinateur. Il m'a permis de découvrir la faisabilité d'une application relativement

simple mais qui m'a demandé beaucoup de tests et d'apprentissage de la bibliothèque OpenCV. J'ai vraiment beaucoup aimé ce domaine, et peut-être aurai-je l'occasion dans un autre projet d'aller plus loin encore.

References

[LAU2020] Laurent Berger, *Traitement d'images et de vidéos avec OpenCV, D-Booker Editions, 2020*

[FAU2023] Julien Faujanet, *OpenCV avec Python, 2023*